ODEs, PDEs, and Fourier Transforms!

PHYS 250 (Autumn 2018) – Lecture 11

David Miller

Department of Physics and the Enrico Fermi Institute
University of Chicago

November 8, 2018

Outline

Reminders from last time

Looked at two primary and exemplary methods for root finding, which is part of the foundation of optimization and differential equation solving.

Fundamental root finding methods

- **Bisection method (aka "incremental search"):**
  - **PROs:** exceptionally simple and requires no knowledge of the function whose roots are sought
  - **CONs:** doesn't use the potentially very useful knowledge of the roots that are sought
- **Newton's Method:**
  - **PROs:** converges much faster than bisection
  - **CONs:** requires a calculation or estimation of the first derivative of the function

Today, we will expand on these algorithms and go several steps further.

Outline

Recall the description of Newton's method

Recall that Newton's method uses the Taylor expansion

$$F(x_0) = F(x+\delta) \approx F(x) + \delta F'(x) + \frac{1}{2}\delta^2 F''(x) + \mathcal{O}(\delta^3) \tag{1}$$

to inform the use of a linear approximation $\delta \approx \Delta$ where

$$\Delta = -\frac{F(x)}{F'(x)} \tag{2}$$

That gives way to an iterative approach that updates the estimate of the position of the root of $F(x)$ as being at x_{i+1}:

$$x_{i+1} = x_i - \frac{F(x_i)}{F'(x_i)} \tag{3}$$

The iteration stops after j iterations when

$$|x - x_j| \leq \epsilon \tag{4}$$

Precision of Newton's method

For any estimate x_i of the method, the error, E_i is the difference between the true root x and the estimate:

$$E_i = x - x_i \tag{5}$$

Merely by inspecting the design of Newton's method, you can see that the precision of the estimate for a subsequent iteration will be given by

$$E_{i+1} = E_i + \frac{F(x)}{F'(x)} \tag{6}$$

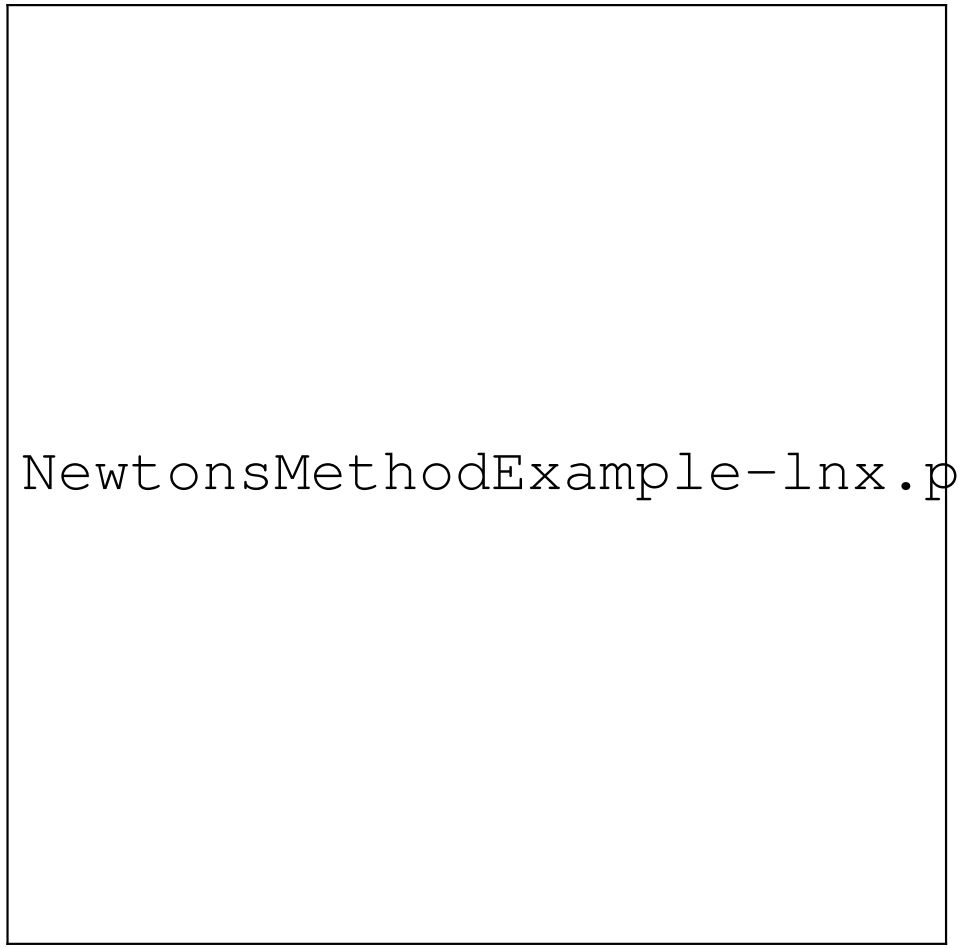
$$= -\frac{F''(x)}{2F'(x)} E_i^2 \tag{7}$$

Convergence of Newton's method

Consequently, Newton's method **converges quadratically**

- the error is the square of the error in the previous step)
- the number of significant figures is roughly doubled in every iteration, provided that x_i is **sufficiently** close to the root.

However, a critical assumption is that $F'(x) \neq 0$; for all $x \in I$, where I is the interval $[x - r, x + r]$ for some $r \geq |x - x_0|$ and x is the true root and x_0 was the starting point.



Pathologies and divergent scenarios

That is definitely not always the case. Let's look at a pathological example. Here is a fun mystery function that I cooked up (since you need to do something similar on your homework):

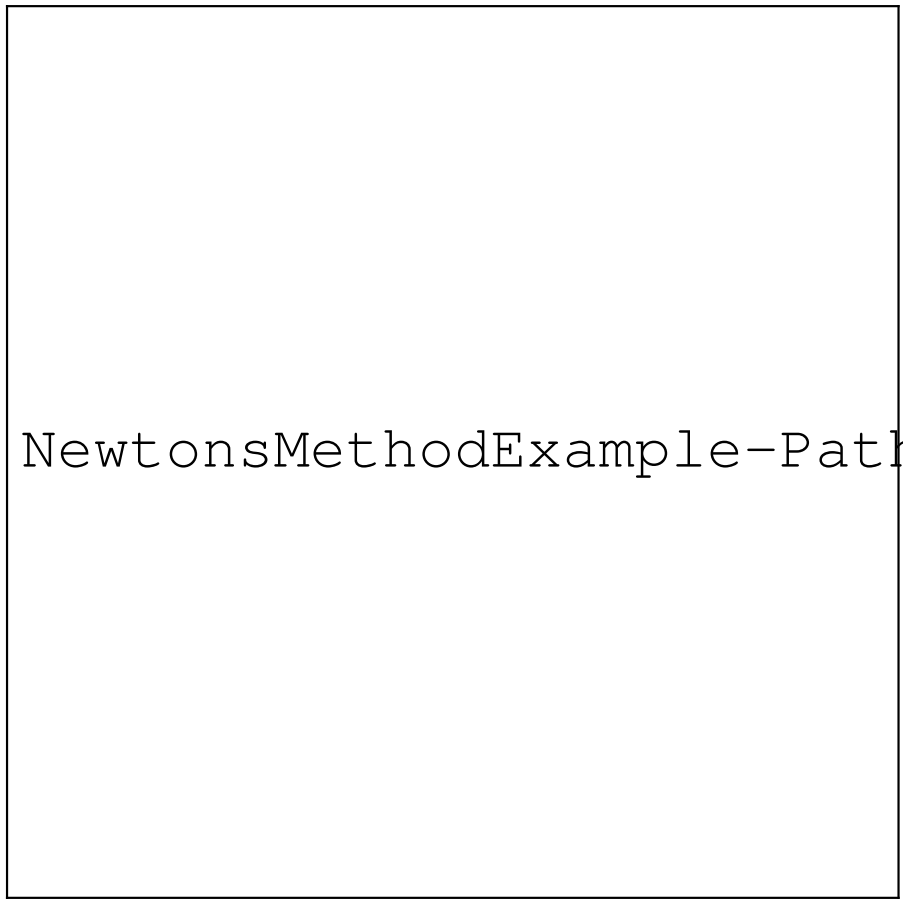
- $x_0 = 2.000$, **7 iterations**
- $x_0 = -2.000$, **2 iterations**
- $x_0 = 3.000$, **2 iterations**
- $x_0 = -1.214$, **4 iterations**
- Slight modification: $x_0 = -1.213$, **no convergence**
- Slight modification: $x_0 = 3.000$, **no convergence**

NewtonsMethodExample-Patho

Pathologies and divergent scenarios

That is definitely not always the case. Let's look at a pathological example. Here is a fun mystery function that I cooked up (since you need to do something similar on your homework):

- $x_0 = 2.000$, **7 iterations**
- $x_0 = -2.000$, **2 iterations**
- $x_0 = 3.000$, **2 iterations**
- $x_0 = -1.214$, **4 iterations**
- Slight modification: $x_0 = -1.213$, **no convergence**
- Slight modification: $x_0 = 3.000$, **no convergence**



Pathologies and divergent scenarios

That is definitely not always the case. Let's look at a pathological example. Here is a fun mystery function that I cooked up (since you need to do something similar on your homework):

- $x_0 = 2.000$, **7 iterations**
- $x_0 = -2.000$, **2 iterations**
- $x_0 = 3.000$, **2 iterations**
- $x_0 = -1.214$, **4 iterations**
- Slight modification: $x_0 = -1.213$, **no convergence**
- Slight modification: $x_0 = 3.000$, **no convergence**

NewtonsMethodExample-Patho

Pathologies and divergent scenarios

That is definitely not always the case. Let's look at a pathological example. Here is a fun mystery function that I cooked up (since you need to do something similar on your homework):

- $x_0 = 2.000$, **7 iterations**
- $x_0 = -2.000$, **2 iterations**
- $x_0 = 3.000$, **2 iterations**
- $x_0 = -1.214$, **4 iterations**
- Slight modification: $x_0 = -1.213$, **no convergence**
- Slight modification: $x_0 = 3.000$, **no convergence**

NewtonsMethodExample-Patho

Pathologies and divergent scenarios

That is definitely not always the case. Let's look at a pathological example. Here is a fun mystery function that I cooked up (since you need to do something similar on your homework):

- $x_0 = 2.000$, **7 iterations**
- $x_0 = -2.000$, **2 iterations**
- $x_0 = 3.000$, **2 iterations**
- $x_0 = -1.214$, **4 iterations**
- Slight modification: $x_0 = -1.213$, **no convergence**
- Slight modification: $x_0 = 3.000$, **no convergence**

NewtonsMethodExample-Patho

Pathologies and divergent scenarios

That is definitely not always the case. Let's look at a pathological example. Here is a fun mystery function that I cooked up (since you need to do something similar on your homework):

- $x_0 = 2.000$, **7 iterations**
- $x_0 = -2.000$, **2 iterations**
- $x_0 = 3.000$, **2 iterations**
- $x_0 = -1.214$, **4 iterations**
- Slight modification: $x_0 = -1.213$, **no convergence**
- Slight modification: $x_0 = 3.000$, **no convergence**

NewtonsMethodExample-Patho

Backtracking

In the last examples above we have a case where the search falls into the pathology of a situation where the initial guess was not **sufficiently close** to the root. an "infinite" loop without ever getting there.

A solution to this problem is called **backtracking**.

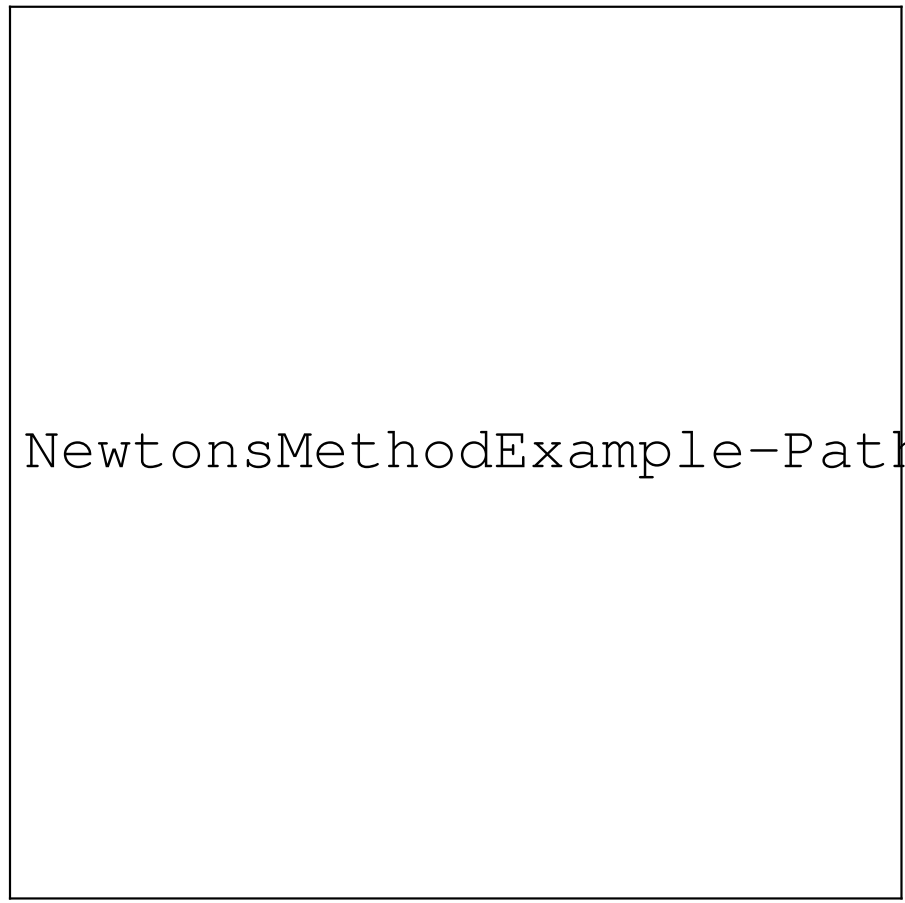
Backtracking

In cases where the new guess $x_0 + \Delta x$ leads to an increase in the magnitude of the function, $|f(x_0 + \Delta x)|^2 > |f(x_0)|^2$, you should backtrack somewhat and try a smaller guess, say, $x_0 + \Delta x/2$. If the magnitude of f still increases, then you just need to backtrack some more, say, by trying $x_0 + \Delta x/4$ as your next guess, and so forth.

Pathological case fixed with backtracking

Fixing the pathological example with backtracking:

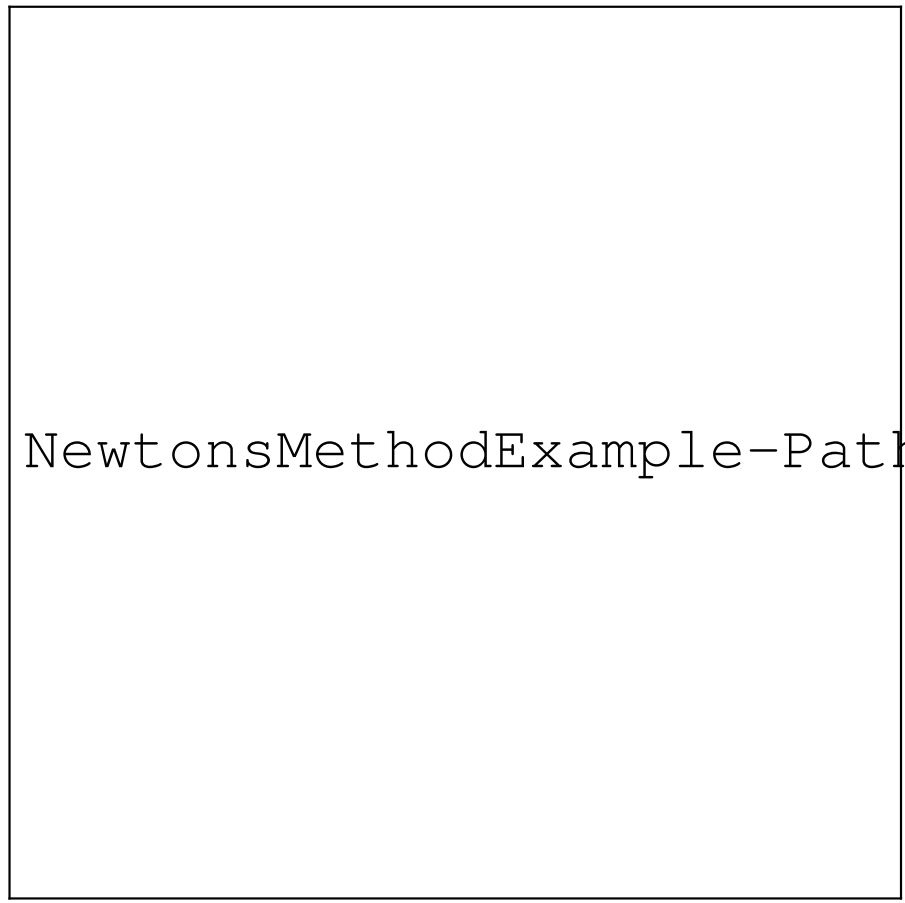
- $x_0 = -1.213$, **no convergence**
- $x_0 = 3.000$, **no convergence**
- $x_0 = 1.500$, **3 iterations**



Pathological case fixed with backtracking

Fixing the pathological example with backtracking:

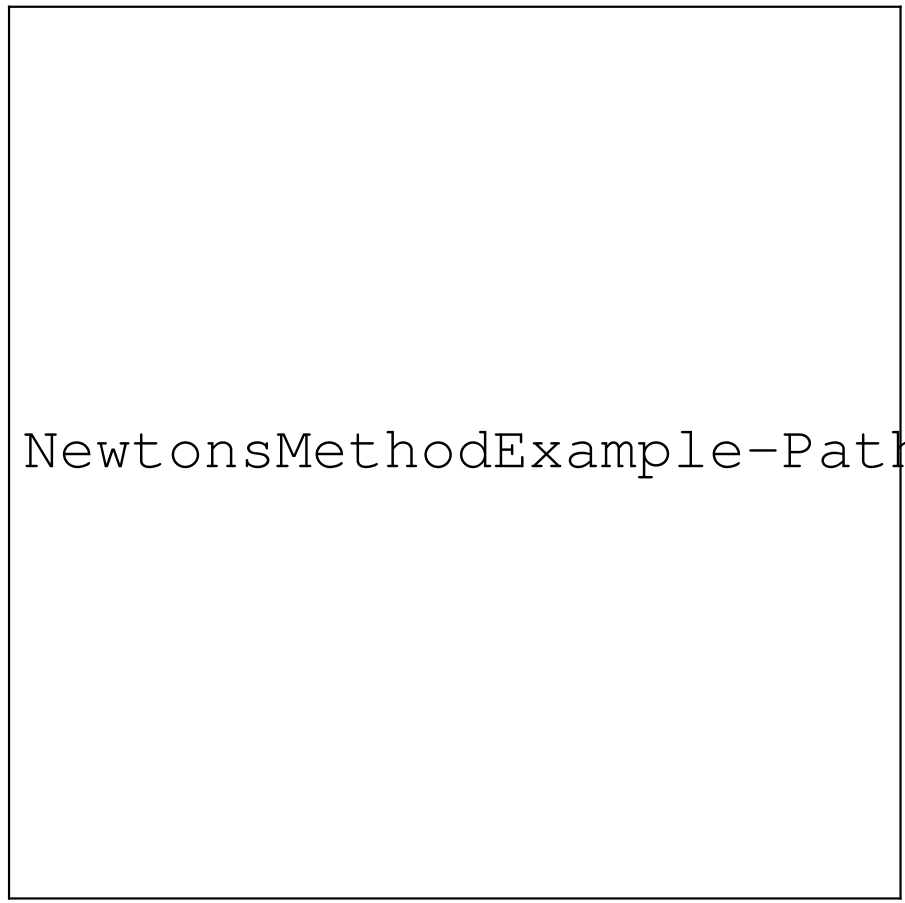
- $x_0 = -1.213$, **no convergence**
- $x_0 = 3.000$, **no convergence**
- $x_0 = 1.500$, **3 iterations**



Pathological case fixed with backtracking

Fixing the pathological example with backtracking:

- $x_0 = -1.213$, **no convergence**
- $x_0 = 3.000$, **no convergence**
- $x_0 = 1.500$, **3 iterations**



Multidimensional problems

Up to this point, we have confined our attention to solving the single equation $F(x) = 0$. Let us now consider the n-dimensional version of the same problem, namely

$$\vec{F}(\vec{x}) = 0 \tag{8}$$

where we allow for a vector of functions $\vec{F} = \{f_1(\vec{x}), f_2(\vec{x}), \ldots, f_n(\vec{x})\}$, and $\vec{x} = \{x_1, x_2, \ldots, x_n\}$.

The solution of n simultaneous, nonlinear equations is a much more formidable task than finding the root of a single equation. The trouble is the lack of a reliable method for bracketing the solution vector $\vec{x}$. Therefore, we cannot always provide the solution algorithm with a good starting value of x, unless such a value is suggested by the physics of the problem.

Newton's method is the workhorse here!

Reminder of the general problem

Start by considering each one of the n functions, $f_n(x)$, separately:

$$f_i(\vec{x}) = f_i(\vec{a}) + \sum_j^n \frac{\partial f_i}{\partial x_j}|_{\vec{x}} \Delta x_j + \frac{1}{2!} \sum_j^n \sum_k^n \frac{\partial^2 f_i}{\partial_j \partial_k}|_{\vec{x}} \Delta x_j \Delta x_k \quad (9)$$

$$= f_i(\vec{a}) + \vec{\nabla} f_i(\vec{a}) \cdot \vec{\Delta x} + \frac{1}{2!} \vec{\Delta x}^{\mathrm{T}} \mathbf{H}(\vec{a}) \vec{\Delta x} \quad (10)$$

where $\mathbf{H}$ is the **Hessian matrix**, describing the **curvature** of $f_i(\vec{x})$ by

$$\mathbf{H}_{j,k} = \frac{\partial^2 f(\vec{a})}{\partial x_j \partial x_k} \quad (11)$$

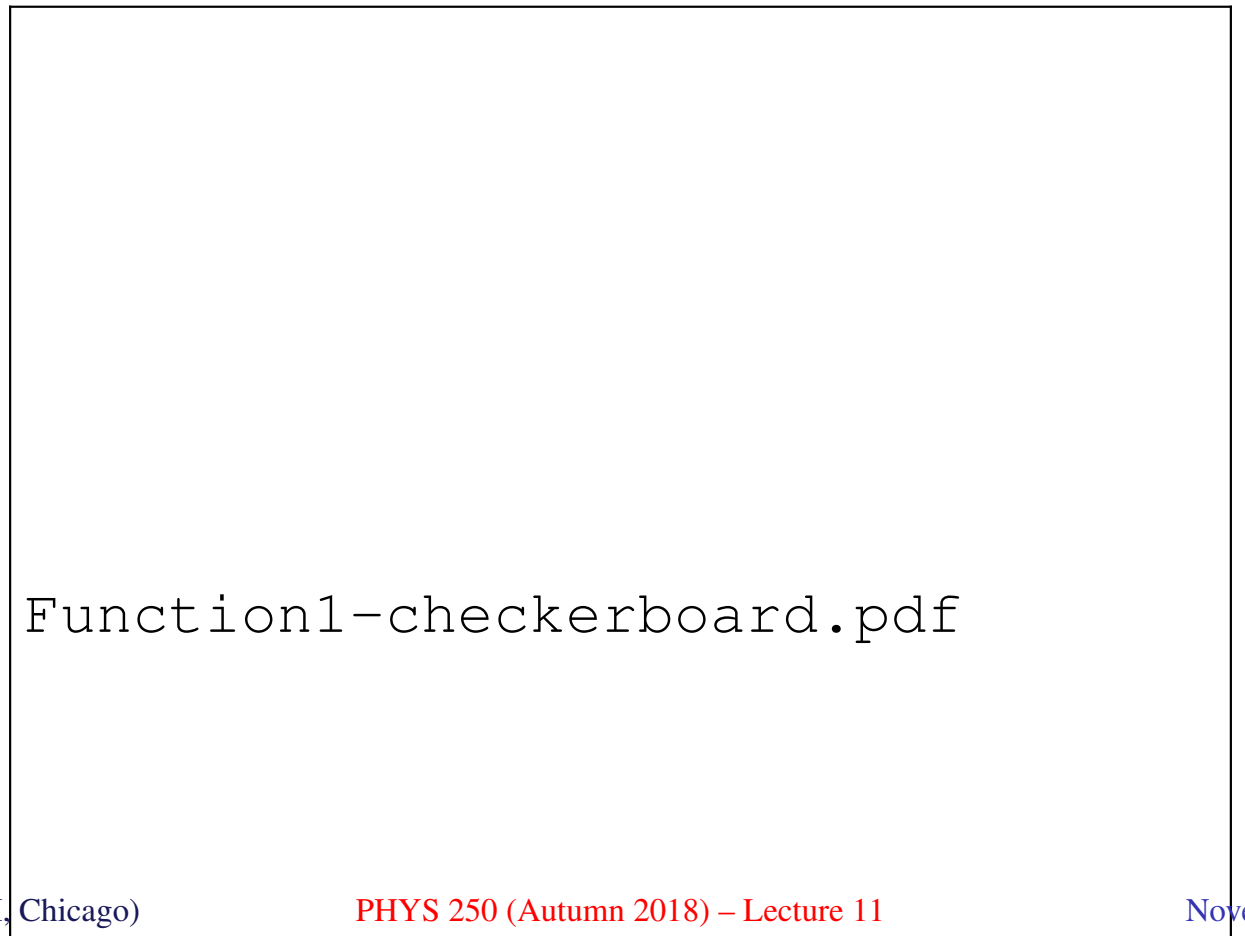
(The determinant of $\mathbf{H}$ is also sometimes referred to as **the Hessian**) and $\vec{\Delta x}$ is a vector that describes the distance from the point about which we are expanding the function $f_i(\vec{x})$.

Specific example (I)

Let's take a very explicit example of a function:

$$f(\vec{x}) = f(x,y) = \ln\sqrt{x^2+y^2} \tag{12}$$

and compute the gradient and Hessian matrix at $x = -2, y = 1$ (see point on graph below).



Specific example (II)

First, we calculate the gradient:

$$\frac{\partial f}{\partial x} = \frac{1}{\sqrt{x^2+y^2}}\left(\frac{1}{2}\frac{2x}{\sqrt{x^2+y^2}}\right) \tag{13}$$

$$= \frac{x}{x^2+y^2} \tag{14}$$

$$\frac{\partial f}{\partial y} = \frac{y}{x^2+y^2} \tag{15}$$

Therefore

$$\vec{\nabla} f(x,y) = \left(\frac{x}{x^2+y^2}, \frac{y}{x^2+y^2}\right) \tag{16}$$

Or, in matrix notation

$$\vec{\nabla} f(x,y) = \begin{bmatrix} \frac{x}{x^2+y^2} \\ \frac{y}{x^2+y^2} \end{bmatrix}, \vec{\nabla} f(-2,1) = \begin{bmatrix} -0.4 \\ 0.2 \end{bmatrix} \tag{17}$$

Specific example (III)

Function1-plane.png

Specific example (IV)

Next, we calculate the Hessian matrix:

$$\frac{\partial^2 f}{\partial x^2} = \frac{(x^2+y^2)-x(2x)}{(x^2+y^2)^2} \tag{18}$$

$$= \frac{-x^2+y^2}{(x^2+y^2)^2} \tag{19}$$

$$\frac{\partial^2 f}{\partial y^2} = \frac{x^2-y^2}{(x^2+y^2)^2} \tag{20}$$

$$\frac{\partial^2 f}{\partial x \partial y} = \frac{\partial^2 f}{\partial x \partial y} = \frac{-2xy}{(x^2+y^2)^2} \tag{21}$$

Therefore

$$\mathbf{H} = \begin{bmatrix} -x^2+y^2 & -2xy \\ -2xy & x^2-y^2 \end{bmatrix} \frac{1}{(x^2+y^2)^2}, \mathbf{H} = \begin{bmatrix} -0.12 & 0.16 \\ 0.16 & 0.12 \end{bmatrix} \tag{22}$$

Jacobian matrix

In the case that the function F is a vector of functions, $\vec{F} = \{f_1(\vec{x}), f_2(\vec{x}), ..., f_n(\vec{x})\}$ (i.e. a system of equations) then the gradient **also has a name: Jacobian**.

$$\mathbf{J} = \begin{bmatrix} \frac{\partial \vec{f}}{\partial x_1} & \cdots & \frac{\partial \vec{f}}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \tag{23}$$

$$\mathbf{J}_{ij} = \frac{\partial f_i}{\partial x_j} \tag{24}$$

The Jacobian matrix is important because if the function $\vec{f}$ is differentiable at a point $\vec{x}$, then $\mathbf{J}$ defines a linear map which is the best (pointwise) linear approximation of the function $\vec{f}$ near the point $\vec{x}$.

That's exactly what we want!

Newton's method for a system of equations

The following steps constitute Newton's method for simultaneous, nonlinear equations:

1. Estimate the solution vector $\vec{x}$
2. Evaluate $\vec{f}(\vec{x})$
3. Compute the Jacobian matrix $\mathbf{J}$ (J_{ij})
4. Setup the simultaneous equations $\mathbf{J}(\vec{x})\vec{\Delta x} = -\vec{f}(\vec{x})$ and solve for $\vec{x}$
5. Let $\vec{x} \leftarrow \vec{x} + \vec{\Delta x}$ and repeat steps 2-5

Comments on matrix computing and manipulations

Physical systems are often modeled by systems of simultaneous equations, and it is very convenient to write these matrix form. In fact, several physical systems can be expressed this way very conveniently.

- Physical optics (lenses, mirrors, etc)
- Electromagnetic waves propagating near boundaries and in matter
- Quantum mechanical systems
- Classical mechanical dynamical systems
- etc, etc

As the models are made more realistic, the matrices often become large, and comput- ers become an excellent tool for solving such problems.

Example: weights and strings (I)

WeightsStrings.pdf

Example: weights and strings (II)

The first 5 equations implement the geometric constraints that the horizontal length of the structure is L and that the strings begin and end at the same height.

$$L_1 \cos\theta_1 + L_2 \cos\theta_2 + L_3 \cos\theta_3 = L \quad (25)$$

$$L_1 \sin\theta_1 + L_2 \sin\theta_2 - L_3 \sin\theta_3 = 0 \quad (26)$$

$$\sin^2\theta_1 + \cos^2\theta_1 = 1 \quad (27)$$

$$\sin^2\theta_2 + \cos^2\theta_2 = 1 \quad (28)$$

$$\sin^2\theta_3 + \cos^2\theta_3 = 1 \quad (29)$$

The last three equations include trigonometric identities as independent equations because we are treating $\sin\theta$ and $\cos\theta$ as independent variables; this makes the search procedure easier to implement.

Example: weights and strings (III)

The basics physics says that since there are no accelerations, the sum of the forces in the horizontal and vertical directions must equal zero.

$$T_1 \sin\theta_1 - T_2 \sin\theta_2 - W_1 = 0 \quad (30)$$

$$T_1 \cos\theta_1 - T_2 \cos\theta_2 = 0 \quad (31)$$

$$T_2 \sin\theta_2 + T_3 \sin\theta_3 - W_2 = 0 \quad (32)$$

$$T_2 \cos\theta_2 - T_3 \cos\theta_3 = 0 \quad (33)$$

Here W_i is the weight of mass i and T_i is the tension in string i. Note that since we do not have a rigid structure, we cannot assume the equilibrium of torques.

These equations represent nine simultaneous nonlinear equations. While linear equations can be solved directly, nonlinear equations cannot.

Example: weights and strings (IV)

We apply to our set the same Newton's method algorithm as used to solve a single equation by renaming the nine unknown angles and tensions as the components of our vector $\vec{x}$ and placing the variables together as a vector:

$$\vec{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} = \begin{bmatrix} \sin\theta_1 \\ \sin\theta_2 \\ \sin\theta_3 \\ \cos\theta_1 \\ \cos\theta_2 \\ \cos\theta_3 \\ T_1 \\ T_2 \\ T_3 \end{bmatrix}$$

Example: weights and strings (V)

The nine equations to be solved are written in a general form with zeros on the right-hand sides and placed in a vector:

$$\vec{f}(\vec{x}) = \begin{bmatrix} f_1(\vec{x}) \\ f_2(\vec{x}) \\ f_3(\vec{x}) \\ f_4(\vec{x}) \\ f_5(\vec{x}) \\ f_6(\vec{x}) \\ f_7(\vec{x}) \\ f_8(\vec{x}) \\ f_9(\vec{x}) \end{bmatrix} = \begin{bmatrix} L_1 x_4 + L_2 x_5 + L_3 x_6 - 8 \\ L_1 x_1 + L_2 x_2 - L_3 x_3 \\ x_7 x_1 - x_8 x_2 - W_1 \\ x_7 x_4 - x_8 x_5 \\ x_8 x_2 + x_9 x_3 - W_2 \\ x_8 x_5 - x_9 x_6 \\ x_1^2 + x_4^2 - 1 \\ x_2^2 + x_5^2 - 1 \\ x_3^2 + x_6^2 \end{bmatrix} = \begin{bmatrix} 3x_4 + 4x_5 + 4x_6 - 8 \\ 3x_1 + 4x_2 - 4x_3 \\ x_7 x_1 - x_8 x_2 - 10 \\ x_7 x_4 - x_8 x_5 \\ x_8 x_2 + x_9 x_3 - 20 \\ x_8 x_5 - x_9 x_6 \\ x_1^2 + x_4^2 - 1 \\ x_2^2 + x_5^2 - 1 \\ x_3^2 + x_6^2 \end{bmatrix}$$

And this is then solved by computing the Jacobian and using Newton's method as usual!

Finite forward difference approximation (I)

Numerical differentiation is related to interpolation: one means of finding the derivative is to approximate the function locally by a polynomial and then differentiate it. I am not going to go into detail about that, since, well, that's about the sum of it.

What we will discuss in a bit more detail are the **finite difference approximations**, which, again, depend on your favorite series, **the Taylor series**. Writing down the Taylor series' for a few definitions of the expansion parameter h, we have:

$$\begin{aligned} f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f''''(x) + \cdots \\ f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f''''(x) - \cdots \end{aligned}$$

This leads to your favorite definition:

$$f'(x) = \frac{f(x+h) - f(x)}{h} - \frac{h}{2}f''(x) - \frac{h^2}{6}f'''(x) - \frac{h^3}{4!}f''''(x) + \cdots$$

Finite forward difference approximation (II)

If we keep only the first terms, then we have the approximation

$$f'(x) = \frac{f(x+h) - f(x)}{h} + \mathcal{O}(h) \tag{34}$$

where the truncation error is $\mathcal{O}(h)$.

Can we do better?

Finite central difference approximation (I)

Yes!, we can do significantly better than the **finite forward difference approximation** with just a little bit of cleverness.

Let's write down the definitions of the Taylor series' again for a few more definitions of the expansion parameter h:

$$
\begin{aligned}
f(x+h) &= f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f''''(x) + \cdots \\
f(x-h) &= f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f''''(x) - \cdots \\
f(x+2h) &= f(x) + 2hf'(x) + \frac{(2h)^2}{2!}f''(x) + \frac{(2h)^3}{3!}f'''(x) + \frac{(2h)^4}{4!}f''''(x) + \cdots \\
f(x-2h) &= f(x) - 2hf'(x) + \frac{(2h)^2}{2!}f''(x) - \frac{(2h)^3}{3!}f'''(x) + \frac{(2h)^4}{4!}f''''(x) - \cdots
\end{aligned}
$$

Finite central difference approximation (II)

Putting this together we can form useful sums and differences:

$$
\begin{aligned}
f(x+h)+f(x-h) &= 2f(x)+h^2 f''(x)+\frac{h^4}{12}f''''(x)+\cdots \\
f(x+h)-f(x-h) &= 2hf'(x)+\frac{h^3}{3}f'''(x)+\cdots \\
f(x+2h)+f(x-2h) &= 2f(x)+4h^2 f''(x)+\frac{4h^4}{3}f''''(x)+\cdots \\
f(x+2h)-f(x-2h) &= 4hf'(x)+\frac{8h^3}{3}f'''(x)+\cdots
\end{aligned}
$$

Note that the sums contain only even derivatives, whereas the differences retain just the odd derivatives, and **the error is at most $\mathcal{O}(h^2)$, and is $\mathcal{O}(h^3)$ for odd powers (i.e. differences)**.

These equations can be viewed as simultaneous equations that can be solved for various derivatives of $f(x)$. The number of equations involved and the number of terms kept in each equation depend on the order of the derivative and the desired degree of accuracy.

Outline

General form and structure of ODEs

The general form of a first-order differential equation is

$$y' = f(x, y) \tag{35}$$

where $y' = \dfrac{dy}{dx}$ and $f(x, y)$ is a given function. The solution of this equation contains an arbitrary constant (the constant of integration). To find this constant, we must know a point on the solution curve; that is, y must be specified at some value of x, say, at $x = a$. We write this condition as

$$y(a) = \alpha \tag{36}$$

Any ordinary differential equation of order n

$$y^{(n)} = f(x, y, y', \ldots, y^{(n-1)}) \tag{37}$$

can always be transformed into n first-order equations. Using the notation

$$y_0 = y,\ y_1 = y',\ y_2 = y'', \cdots, y_{n-1} = y^{(n-1)} \tag{38}$$

and the equivalent first-order equations are

$$y_0' = y_1,\ y_1' = y_2,\ y_2' = y_3, \cdots, y_n' = f(x, y_0, y_1, \ldots, y_{n-1}) \tag{39}$$

Initial vs. boundary value problems

The solution now requires the knowledge of n conditions to fully specify. If these conditions are specified at **the same value of x**, the problem is **an initial value problem**. Then the **initial conditions**, have the form:

$$\begin{aligned} y_0(a) &= \alpha_0 \\ y_1(a) &= \alpha_1 \\ &\vdots \\ y_{n-1}(a) &= \alpha_{n-1} \end{aligned}$$

On the other hand, if y_i are specified **at different values of x,** the problem is a **boundary value problem**.

$$\begin{aligned} y'' &= -y \\ y(0) &= 1 \\ y(\pi) &= 0 \end{aligned}$$

Recall the Taylor series approach generally

Writing the conditions for the initial value problem from the previous slides as

$$\vec{y}' = \vec{F}(x, \vec{y}), \qquad \vec{y}(a) = \vec{\alpha} \tag{40}$$

where

$$F(x, \vec{y}) = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ f(x, \vec{y}) \end{bmatrix}, \tag{41}$$

We can sort of "brute force" our way through it by repeatedly computing derivatives numerically using Newton's method. That's perfectly valid, but there are also better ways to go about it.

Avoiding repeated differentiation: Runge-Kutta

The aim of Runge-Kutta methods is to eliminate the need for repeated differentiation of the differential equations. Because no such differentiation is involved in the **first-order Taylor series** expression:

$$\vec{y}(x+h) = \vec{y}(x) + \vec{y}'(x)h = \vec{y}(x) + \vec{F}(x,\vec{y})h \tag{42}$$

This first-order version is referred to as **Euler's method (aka Euler's Rule)**.

Effectively, what this is doing is to start with the known initial value of the dependent variable, $y_0 \equiv y(x=0)$, and then use the derivative function $f(x,y)$ to find an approximate value for y at a small step $x = h$ forward in time; that is, $y(x=h) \equiv y_1$.

We know from our discussion of differentiation that the error in the forward-difference algorithm is $\mathcal{O}(h)$, and so this too is the error in Euler's rule.

Second-order Runge-Kutta algorithm

The Runge-Kutta algorithms for integrating a differential equation are based upon the formal (exact) integral of our differential equation:

$$\frac{dy}{dx} = f(x,y) \Rightarrow y(x) = \int f(x,y)dx \tag{43}$$

And therefore

$$y_{n+1} = y_n + \int_{x_n}^{x_{n+1}} f(x,y)dx \tag{44}$$

The key insight is to expand $f(x,y)$ in a Taylor series about the **midpoint of the integration interval** and retain two terms:

$$f(x,y) \simeq f(x_{n+1/2}, y_{n+1/2}) + (x - x_{n+1/2})\frac{df}{dx}(x_{n+1/2}) + \mathcal{O}(h^2) \tag{45}$$

As you recall from the **finite central difference** algorithm, only **odd powers** of h remain, and thus when used inside the integral above, the terms with $(x - x_{n+1/2})^{n\in\text{odd}}$ vanish. We are left with

$$y_{n+1} \simeq y_n + hf(x_{n+1/2}, y_{n+1/2}) + \mathcal{O}(h^3) \tag{46}$$

Difficulty with the second-order Runge-Kutta algorithm

The price for improved precision is having to evaluate the derivative function and y at the middle of the interval, $x = x_n + h/2$.

And there's the rub: we don't know the value of $y_{n+1/2}$ and cannot use this algorithm to determine it.

The way out of this issue is to use Euler's algorithm for $y_{n+1/2}$:

$$y(x + h/2) = y_n + \frac{1}{2}h\vec{y}' = y_n + \frac{1}{2}hf(x_n, y_n) \tag{47}$$

In this way, the known derivative function f is evaluated at the ends and the midpoint of the interval, but that only the (known) initial value of the dependent variable y is required. This makes the algorithm self-starting.

$$y_{n+1} \simeq y_n + k_2 \tag{48}$$

where

$$k_2 = h\vec{f}(x_n + \frac{h}{2}, \vec{y}_n + \frac{k_1}{2}), \qquad k_1 = h\vec{f}(x_n, y_n) \tag{49}$$

Precision

As discussed, the second order Runge-Kutta only achieves an $\mathcal{O}(h^3)$ precision. That precision is quickly insufficient for many applications.

The fourth-order Runge-Kutta method achieves an $\mathcal{O}(h^4)$ precision by approximating y as a Taylor series up to h^2 (a parabola) at the midpoint of the interval.

This approximation provides an excellent balance of power, precision, and programming simplicity. There are now four gradient terms to evaluate with four subroutine calls needed to provide a better approximation to $f(x, y)$ near the midpoint.

Higher order calculations: `rk4`

Without deriving anything, I simply write out the definition and then we'll elaborate on the implications

$$y_{n+1} \simeq y_n + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) \tag{50}$$

where

$$k_1 = h\vec{f}(x_n, y_n) \tag{51}$$

$$k_2 = h\vec{f}\left(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}\right) \tag{52}$$

$$k_3 = h\vec{f}\left(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}\right) \tag{53}$$

$$k_4 = h\vec{f}\left(x_n + h, y_n + k_3\right) \tag{54}$$

The benefit of the calculation of the additional terms is a $\mathcal{O}(h^4)$ precision on the final results.

Outline

The issue with boundary value problems

In an initial value problem we were able to start at the point where the initial values were given and march the solution forward as far as needed.

This technique does not work for boundary value problems, because there are not enough starting conditions available at either endpoint to produce a unique solution.

The simplest two-point boundary value problem is a second-order differential equation with one condition specified at $x = a$ and another one at $x = b$. Here is an example of such a problem:

$$y'' = f(x, y, y'), \qquad y(a) = \alpha, \qquad y(b) = \beta \tag{55}$$

The whole point is to attempt to turn these equations into the initial value problem such that

$$y'' = f(x, y, y'), \qquad y(a) = \alpha, \qquad y' = u \tag{56}$$

and the key to success is finding the correct value of u.

Root finding

Really, this is just a matter of **root finding**!! Which, of course, you now know very well how to do!

The solution of the initial value problem depends on u, the computed value of $y(b)$ is a function of u; that is,

$$y(b) = \theta(u) \tag{57}$$

And hence, u is a root of the expression:

$$r(u) = \theta(u) - \beta = 0 \tag{58}$$

where $r(u)$ is the boundary residual (difference between the computed and specified boundary value at $x = b$)

This is the essence of the shooting method, which we will discuss next time